Towards the Interoperability of Web, Database, and Mass Storage Technologies for Petabyte Archives

Reagan Moore, Richard Marciano, Michael Wan,
Tom Sherwin, Richard Frost
San Diego Supercomputer Center
P.O. Box 85608
San Diego, CA 92186-9784
E-mail: {moore, marciano, mwan, sherwint, frost}@sdsc.edu
Phone: +1-619-534-5073
Fax: +1-619-534-5152

Abstract

At the San Diego Supercomputer Center, a Massive Data Analysis System (MDAS) is being developed to support data-intensive applications that manipulate terabyte-sized data sets. The objective is to support scientific application access to data whether it is located at a Web site, stored as an object in a database, and/or stored in an archival storage system. We are developing a suite of demonstration programs which illustrate how Web, database (DBMS), and archival storage (Mass Storage) technologies can be integrated. An Application Presentation Interface is being designed that integrates data access to all of these sources.

We have developed a data movement interface between the Illustra object-relational database and the NSL UniTree archival storage system running in production mode at the San Diego Supercomputer Center. With this interface, an Illustra client can transparently access data on UniTree under the control of the Illustra DBMS server. The current implementation is based on the creation of a new DBMS storage manager class, and a set of library functions that allow the manipulation and migration of data stored as Illustra *"large objects"*.

We have extended this interface to allow a Web client application to control data movement between its local disk, the Web server, the DBMS Illustra server, and the UniTree Mass Storage environment. This paper describes some of the current approaches for Web, DBMS, and Mass Storage interoperability, and presents a framework for successfully integrating these technologies. This framework is measured against a representative sample of environmental data extracted from the San Diego Bay Environmental Data Repository. Practical lessons are drawn and critical research areas are highlighted.

1. Introduction

A series of projects are being undertaken at the San Diego Supercomputer Center to develop the software technology that is needed to support data-intensive scientific applications (Moore [1]). These projects explore various aspects of distributed data handling capabilities, including integration of object-relational database management systems (ORDBMS) (Moore [2]) with archival storage, development of Web and Java interfaces for databases and archival storage systems, and development of a standard API for accessing data from heterogeneous sources.

The ability to manipulate very large data sets and large collections of data sets is a chief goal of the Massive Data Analysis System (MDAS) project (Moore [3]). Two features are essential components of this system: accessing data sets by attribute rather than UNIX file

name, and transporting very large data sets across parallel I/O channels. Object-relational database technology is used to support query by attribute, and archival storage technology is used to support third-party parallel transfer of data sets. The MDAS project is integrating these technologies to create a data handling environment capable of supporting terabyte-sized data sets. *Large objects* that are controlled by the database are stored in the archive instead of the database local disk. This allows the ORDBMS to manage collections of data objects which exceed the local database disk capacity. By using transportable methods for manipulating data objects, it is also possible to minimize CPU execution constraints. When a query is processed, both the data object and the transportable method are sent to a system on which the analysis is then done. The data handling system effectively serves as a data scheduler, moving data and associated computational methods to available compute resources.

The data-handling system architecture is presented in Figure 1. Three different clients are shown accessing the system, corresponding to interactive Web-based access, scientific application access, and DBMS access to support data movement between multiple data handling systems. The DBMS maintains large objects within the archive and has the ability to schedule computationally intensive work on various production systems. The system is designed to support third-party transfer of data from the archive directly to the requesting system across parallel I/O channels.

To gain insight into issues associated with database/mass-storage integration, we built a prototype using Illustra (Illustra [4]) as the database engine and NSL UniTree as the mass storage system. NSL UniTree is a hierarchical archival storage system currently running in production mode at SDSC. The hardware platform consists of a single IBM RS/6000 99J workstation, a disk cache of 100 GB, two StorageTek tape silos and 8 tape drives transferring data at rates up to 2.9 MB/s with 1 controller per four drives. The system is capable of storing up to 20 TB (terabytes) of data. The most recently accessed files are staged on the large disk cache and the rest are migrated to tape.

A second research prototype has been created through a similar integration of Postgres95 Stonebraker [5, 6]) with NSL UniTree. The database runs on a 17-node IBM SP-2, which controls a 500-GB IBM Serial Storage Architecture (SSA) Disk Subsystem and a 60-TB IBM 3494 Tape Library Dataserver using six high-capacity 3590 Magstar tape drives. This system will be used in collaboration with IBM to develop a Massive Analysis Testbed that integrates the DB2 Parallel Edition DBMS with the High Performance Storage System (HPSS) mass storage system (Archival Storage Research at SDSC [7]). Data transfer rates of 300 MB/sec are expected from this system. The nominal design point for expansion of this testbed is to sustain at least 1 GB/sec data access rate for each additional terabyte of disk. The design point for data access to tape is 1 GB/sec per 100 terabytes storage capacity.

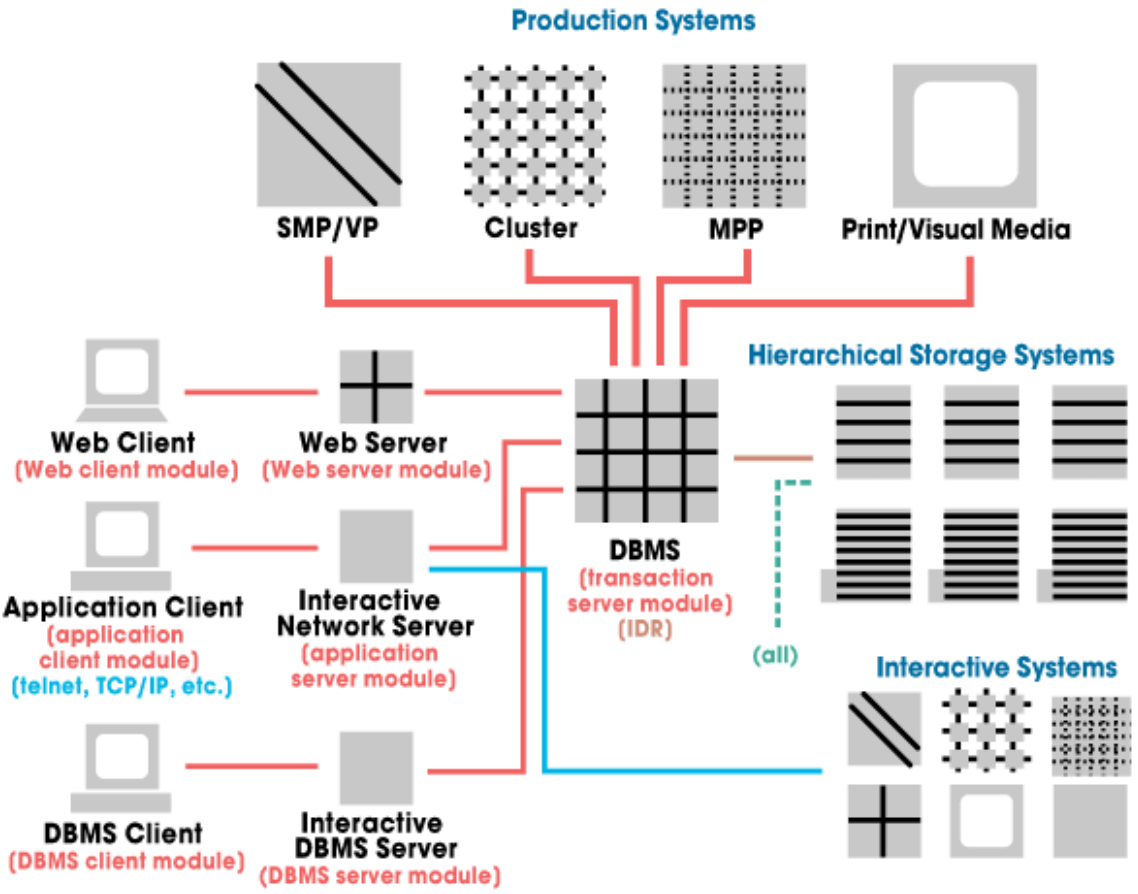


Figure 1: MDAS System Architecture

This paper presents the various software interfaces that have been developed in the research prototypes. The mass storage interface is described in section 2, the database interface in section 3, and a Web interface in section 4. A real-world example consisting of a representative sample of environmental data extracted from the San Diego Bay Environmental Data Repository is shown in section 5, and concluding remarks and expanded data access scenarios are given in section 6.

2. Mass Storage interface (MSI)

We have developed a data movement interface between the Illustra object-relational database and the NSL UniTree archival storage system. With this interface, an Illustra client is able to transparently access data on UniTree through the Illustra server by sending appropriate queries and commands.

2.1 MSI software features

Metadata describing the data set attributes are stored on the local disk under the database control. *Large objects* (data items larger than approximately 8K bytes) are stored in UniTree through the Illustra/UniTree interface. A *large object* is a defined data type that is created using the Illustra DBMS facilities. *Large objects* stored in UniTree have all the database properties of any Illustra object, such as transaction rollback, crash recovery, and

multi-user protection. Unreferenced *large objects* can be removed from the database by issuing the *"vacuum"* SQL statement. However, once created they cannot be overwritten or appended to. Illustra supports a built-in data type for pointing to a large object, called *"large_object"*. When a user selects a *large object* from a table, the returned value is a handle to the *large object*. The handle is a character string, such as *'I098723987211'*, which is used to define a unique data set within the UniTree system or the local database disk.

From an Illustra client standpoint, except for the difference in access speed between local disk and remote archive, *"large objects"* stored in UniTree behave exactly the same as other *"large objects"*. A user can use normal queries and commands to perform the following tasks:

- Store and retrieve *large objects* between local disk and UniTree.
- Vacuum unreferenced *large objects* stored in UniTree.
- *"Dump"*, *"restore"* and *"recover" "large objects"* stored in UniTree.

To test the integration, *"large object"* files stored in UniTree were intentionally deleted after a *"dump"*. *"Restore"* and *"Recover"* were then used to restore the deleted files.

2.2 MSI software implementation

The implementation of the MSI is done by adding a storage type - "UniTree" to the ORDBMS storage manager. This required creating a set of 35 new UniTree specific access functions for operating on data sets. Example functions are *open*, *close*, *read*, *write*, *flush*, *abort*, and *synch*. The design provides a one-to-one counterpart for each UniTree access function with the corresponding function for accessing magnetic disk storage.

Similar to the magnetic disk storage type functions, the UniTree access functions do not make direct I/O calls. Instead they perform I/O through Virtual File Descriptor functions that call the *libnsl.a* and *libnsltree.a* UniTree libraries to interact with the UniTree Mass Storage System. These libraries provide client processes with UNIX-like I/O access functions as well as functions that are specific to UniTree such as file staging and migration.

3. Database software interface

User functions have been developed to allow user-level control over the storage location of the data sets within the integrated database/archival storage system. Note that the data sets might initially be stored on the user's local disk, then stored as a large object on the database system disks, or stored in the archival storage system. The responsiveness of the system typically improves, the closer the cache level is to the user. Hence user control is needed to optimize access performance.

Three *DataBlade* functions - **myFileToLO**(), **LocalToUtree**() and **UtreeToLocal**() have been created to provide an easier way for an Illustra user to convert local files to large objects on UniTree and to migrate objects between UniTree and database file systems. A *DataBlade* is a mechanism to extend the Illustra server to manage new data types and functions on these data types.

The *DataBlade* terminology comes from the following analogy: just like a general purpose utility knife can be extended to perform different cutting jobs by inserting special-purpose blades, so can the Illustra Server be extended to manage new data types by snapping in the required *DataBlade*. Basically, these functions use the *large_object* manipulation functions of Illustra to move *large objects* between database magnetic disk and UniTree.

3.1 Data caching functions

1) myFileToLO (filename, flags, smgr) - used to copy a local disk file to a *large object* stored in the archival storage system. This is the same as the *FileToLO ()* function that comes with Illustra with the exception that a parameter - *smgr* has been added to allow users to specify the storage type for the *large object*.

Filename = The name of the file to be converted to large object.

Flags = the location of the file :
0 = the file is on the client machine.
1 = the file is on the server machine.

Smgr = the storage type where you want to store the large object.
0 = local disk.
2 = UniTree.

The *returned value* is the LO handle of the newly created object.

2) UtreeToLocal(large_object) - used to migrate large objects from archival storage to the database disk.

Large_object = The LO handle of the large object to be migrated.

The *returned value* is the LO handle of the newly created large object.

3) LocalToUtree(large_object) - It is used to migrate large objects from the database disk to archival storage.

Large_object - the LO handle of the large object to be migrated.

The *returned value* is the LO handle of the newly created large object.

3.2 Examples

The following script illustrates the use of these three *DataBlade* functions. One could interactively enter this script using the *msql* command shell. The text in bold corresponds to the system's response. The *large object* handle value encodes the cache location of the data set (I2... means that the *large object* actually resides in the UniTree archival storage system, and I0... means that it is on the Illustra Server disk). Two data sets are stored in the system; "foo1" on UniTree and "foo2" on database disk. "foo1" is then migrated onto database disk, and "foo2" is migrated into UniTree.

```
---- First, create a table named LOTest.
create table LOTest
(
        name text,
        myLO large_object
);
```

```
---- The following command will store the large object in Unitree
insert into LOTest values ('foo1',
                myFileToLO ('file1', 0, 2));
one  row  inserted

---- The following command will store the large object to local
---- disks.
insert into LOTest values ('foo2',
                myFileToLO ('file2', 0, 0));
one  row  inserted

select * from LOTest;
-----------------------------
|name         |myLO            |
-----------------------------
|foo1           |I2105826192499|
|foo2           |I0108798611396|
-----------------------------

----
---- The next 2 commands migrate the large objects between the
---- local disk and UniTree.
----
update LOTest set myLO=UtreeToLocal(myLO) where  name='foo1';
update LOTest set myLO=LocalToUtree(myLO) where  name='foo2';

select * from LOTest;
-----------------------------
|name         |myLO            |
-----------------------------
|foo1           |I0109780388206|
|foo2           |I2100786116526|
-----------------------------

---- Illustra does not delete the old object automatically, so
---- you need to vacuum it.
vacuum from LOTest;
```

4. Web software interface

A Web Server side C-language CGI (Common Gateway Interface) to Illustra was developed. This program allows the user to build or specify existing SQL queries which are then passed to the Illustra server. In essence the C interface program is a multi-purpose program acting as a Web Client program (generating HTML) and also as an Illustra DBMS client program (connecting remotely to the Illustra Server, issuing SQL commands, collecting SQL command result sets, disconnecting from Illustra, extracting information from the result sets and displaying it to the screen). The DBMS client part is done by linking the code to the *libmi.a* Illustra C-programming interface library.

Other than the fact that the SQL commands which are sent to the server allow the use of the new UniTree DataBlade functions, the Web software interface is a standard interface that one would find in most Web to DBMS integrations.

This section illustrates how one might be able to control this integrated environment on a simple web example (section 4.1) and concludes with general considerations on how we dealt with server time-out issues (section 4.2) .

4.1 An integrated example

The WEB demo presented here can be executed from the following Web page:

http://www.sdsc.edu/projects/MassDataAnal/Demo_Illustra+Unitree/

The following explanations are meant to serve as an introductory guide to this web demo.

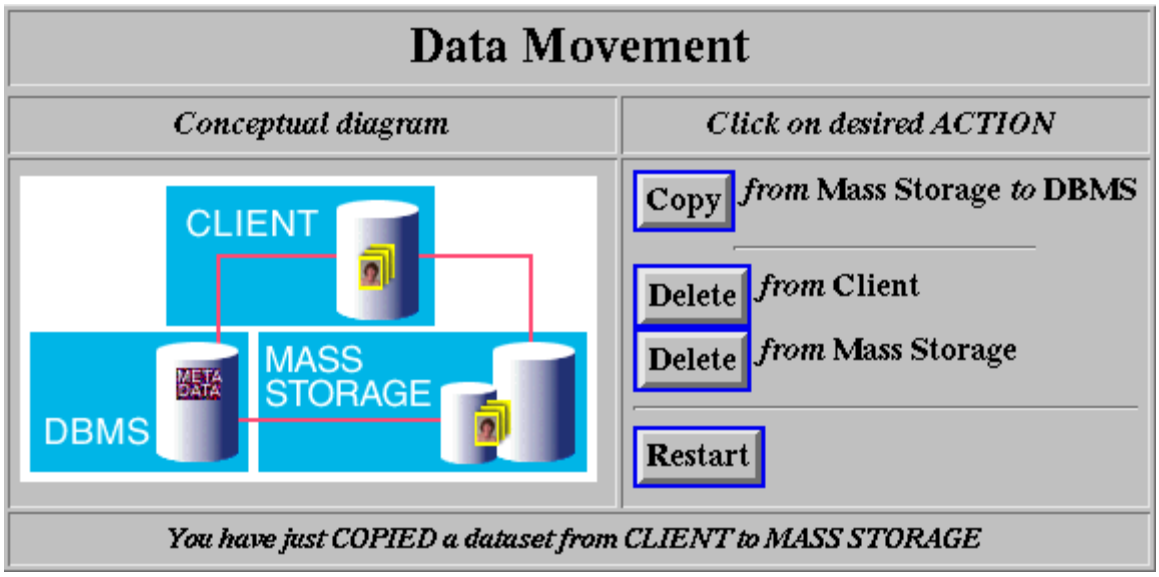


Figure 2: Data Movement Integration demonstration screen

The demo presents the equivalent of a finite state machine. The three states are: Client, DBMS, and Mass Storage. The demo illustrates data movement between a Client (Web client, for example), a DBMS and a Mass Storage environment. Tokens representing data objects are allowed to flow along the connecting arcs and are associated with state boxes. There are two kinds of data tokens: *Metadata* and *Datasets*. A *Dataset* token is a file in this demo and can appear in any of the three states. A *Metadata* token can only appear in the DBMS state and represents the existence of a non-empty Illustra table. The actual table contains two fields including a *large_object* field which is a handle to a *large object* stored either in the DBMS state or in the Mass Storage state.

The initial state of the system indicates that a file resides on the client side. Context-sensitive action buttons allow you to choose the data flow paths of interest. For example, initially, one could load the file into the DBMS and have the file's final destination be on the DBMS machine ("*Copy to DBMS"* ACTION button) or one could load the file into the DBMS but have its final location be on the Mass Storage file (*"Copy to Mass Storage"* ACTION button, as in Figure 2.). In either case a *Metadata* token would appear on the DBMS state box, indicating the existence of a non-empty SQL table.

Allowed actions include *"Copy"*, *"Delete"*, and *"Restart"*. Explanations of what was just carried out appear on the bottom of the diagram as well as detailed instructions of what the Illustra Metadata table's content is and how this operation was carried out. Features including a *"large object display"* section are provided. This allows you to display the contents of the file object directly from its current location (Client, DBMS, or Mass Storage) directly streaming it to the Web browser window without ever going through any kind of intermediate storage. This allows you in particular to verify that the file object has successfully been migrated to its new destination.

While the data movement window is being updated and explanations are being provided, the operations are carried out behind the scenes in real-time. This demo provides a window of observation into the integrated "Web-Database-Mass Storage" environment (Marciano [8]).

4.2 Dealing with time-outs

Time-outs are a delicate issue, given that all three servers (Web, Illustra, UniTree) have their own default time-out thresholds. For example, an unsatisfied Web request will time out after a preset amount of time, generating a message warning you that the server you are trying to connect to might be temporarily unavailable. The following solutions are first-level attempts at dealing with some of the more obvious time-out problems.

Access of *large_objects* stored in UniTree may hang for a long time because of the following two reasons:

1. The UniTree server has died.

2. The *large object* file has migrated to tape. It could take 10 minutes or more to stage a file from tape to disk.

There are at least three scenarios that need to be handled:

1) The Illustra server tries to connect to the UniTree server but the UniTree server is not present. The current UniTree library causes the Illustra server to hang indefinitely or almost indefinitely.

Our **solution** is to make the connection request time out in 30 seconds. An error message is sent to the client when a time-out occurs.

2) The *large object* file has migrated to tape and it may take 10 minutes or more to stage the file from tape to disk. This causes the Illustra server to block until the file is staged.

Our **solution** is to make the open request time out in 2 minutes. A warning message is sent to the Illustra client every 30 seconds to inform the user what has been taking place. When the 120 second time-out is triggered, another error message is sent to the client before failing.

3) The UniTree server dies when the Illustra server is doing read/write operation to and from UniTree. There is a 2 second time-out for read/write in *libnsl.a*. In this case a regular read/write error message is sent.

5. Environmental Data Testing

This section describes how the example interface in section 3.1 was extended to handle real-world data on an existing environmental sciences application.

Please refer to the web location **"http://www.sdsc.edu/~sdbay"** for more information on the San Diego Bay Project, an environmental data repository which contains chemical, physical, and biological data for the bay of San Diego and which can be accessed over the Web through a clickable map of the Bay. Currently, the project uses flat files and only emulates a database engine.

The integration effort has involved porting a representative subset of this environmental data directly to the Illustra database and experimenting with clickable map search scenarios that allow the data to be displayed over the Web and stored both in the local store of the Illustra DBMS as well as on the UniTree mass storage store.

An example of a clickable map search interface for the *Integrated San Diego Bay prototype* that we are developing is shown in Figure 3. After defining the appropriate geographic box, an SQL query would be submitted to the Illustra server. The query would take the user-specified bounding box and intersect it with the list of registered bounding boxes stored as metadata with each image. The image itself is stored as a *large object* that can reside either on the Illustra side or on the NSL UniTree side. To achieve this result we wrote an Illustra SQL user-defined function (UDF) called GIS_overlap():

```
create function GIS_overlap( arrayof( real ), arrayof( real ) )
returns boolean
as external name 'GIS_overlap.so'
language C;
```

Databases such as Illustra provide a special 2D Spatial Data *DataBlade* with more efficient "GIS_overlap"-like functions that one could use directly.

The *clickable map interface* (see Figure 3) allows the viewer to build a running list of locations of interest in the bay and submit those to the search engine, which returns a list of environmental files of interest, broken down into those three categories (physical, chemical, and biological).

Clicking on the file name in RECORD 1 in Figure 4 would display the actual file (see Figure 5). Note here that the file is directly streamed from NSL UniTree to the Web browser window.
As far as the user is concerned, this is fairly transparent, except when a file has been totally migrated off to tape, and longer waiting periods occur. This flexible scheme allows us to choose from a hierarchy where specific data items might reside on the Web Server's local disk, on the DBMS's disk, or all the way out on the Mass Storage area, which itself allows pre-caching on the host RS-6000 disk. We are currently evaluating where to store this environmental data in the storage hierarchy.

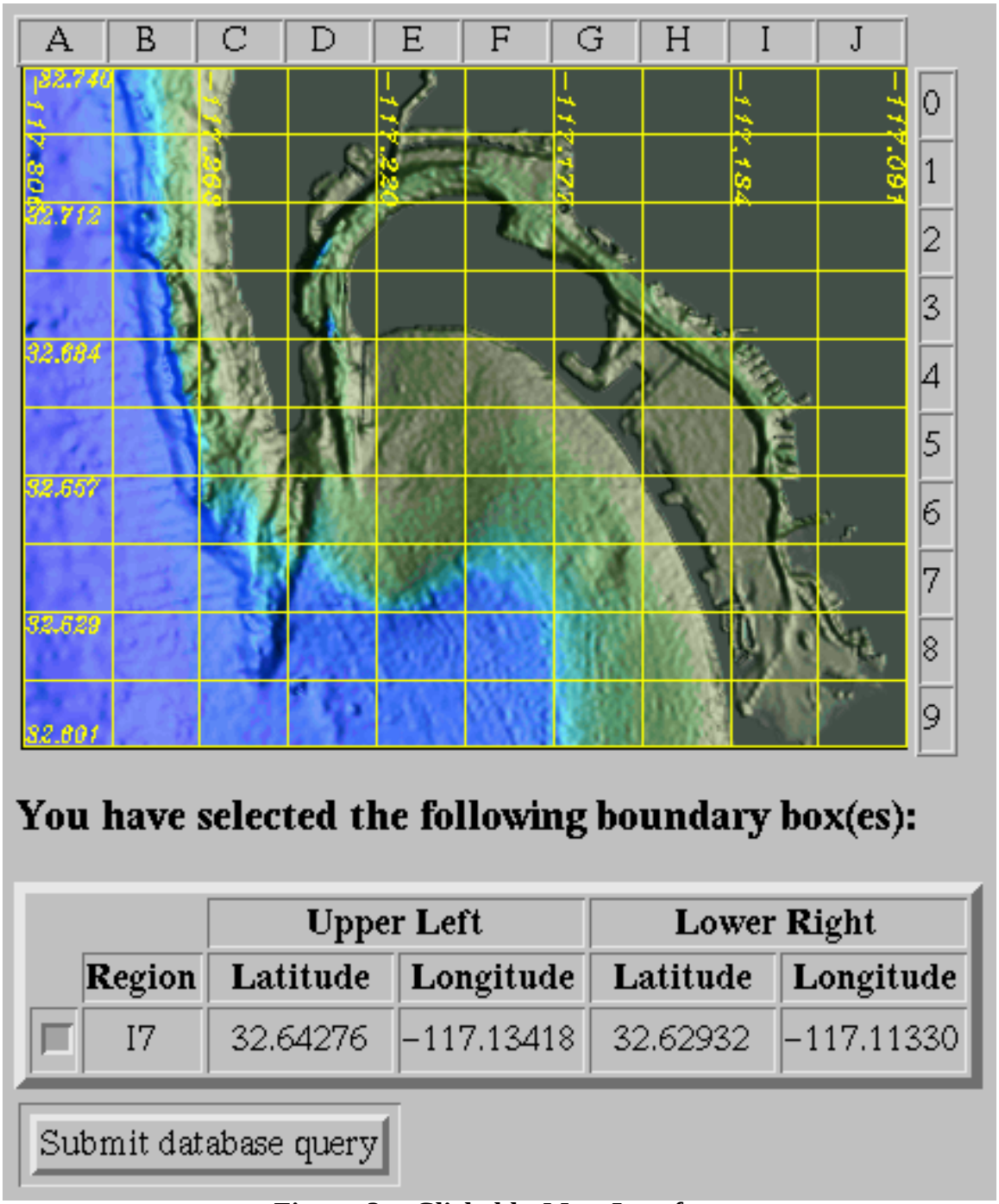


Figure 3: Clickable Map Interface

About to try send the following command to the **Illustra** database.....

```
select f_name, categ, f_text, myLO from sdbay
where GIS_overlap('[32.642761, -117.134178, 32.629322, -117.113304]', bbox);
```

-- STATUS INFO --

Initiating database server connection...

OK: connected to server **pulsar**
OK: connected to database **vlado**
3 rows affected by SELECT command

...Closing database server connection

-- RESULT SET --

RECORD 1

```
sdbay[0].f_name =       eelgrass.raw.txt.dx.00.gif
sdbay[0].categ =        Biological
sdbay[0].f_text =       Eelgrass density in the bay
sdbay[0].f_myLO =       I2101796152168
```

RECORD 2

```
sdbay[1].f_name =       sio-map.gif
sdbay[1].categ =        Physical
sdbay[1].f_text =       Line drawing of San Diego Bay
sdbay[1].f_myLO =       I0102835749996
```

RECORD 3

```
sdbay[2].f_name =       ARSENIC.ppm.txt.gen.930804.00.gif
sdbay[2].categ =        Chemical
sdbay[2].f_text =       Sediment test
sdbay[2].f_myLO =       I2103801418438
```

Figure 4: SQL query results

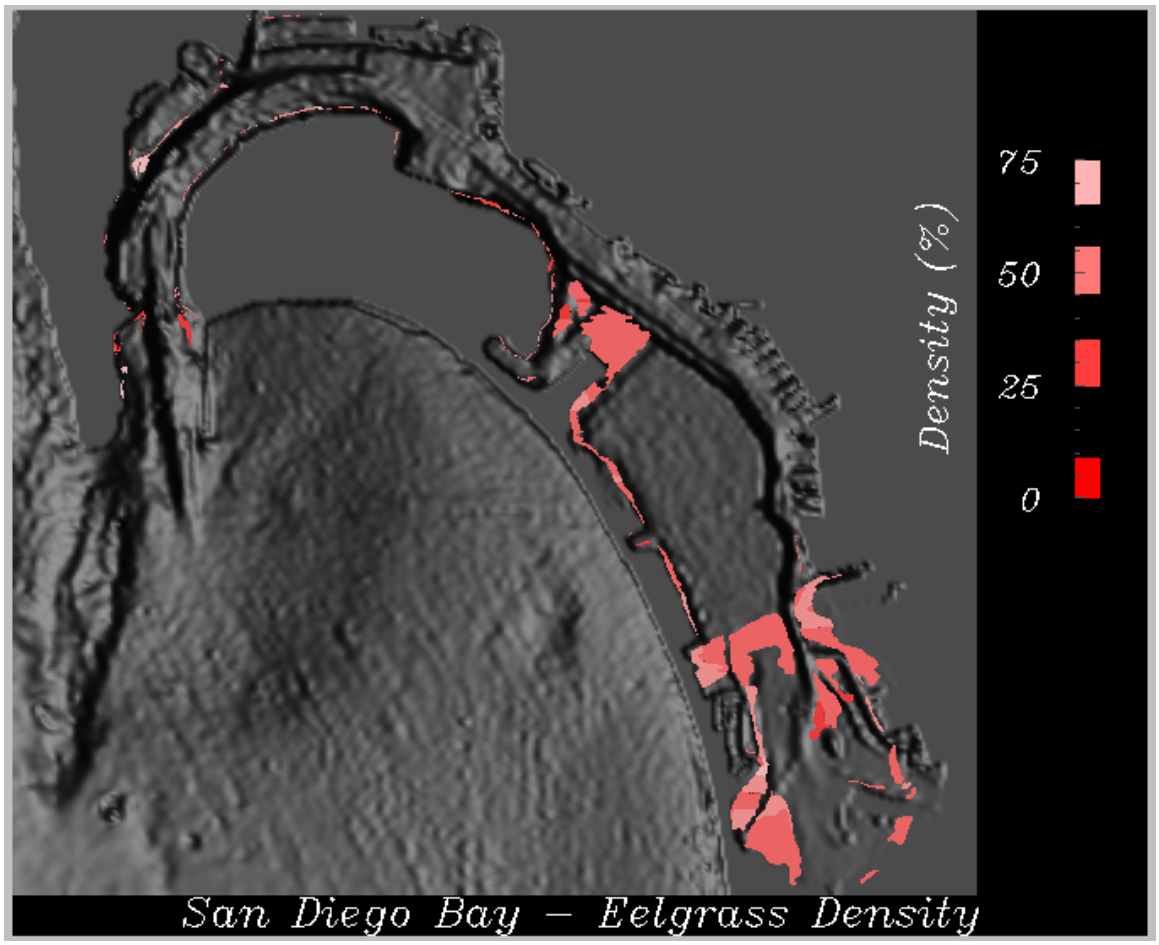


Figure 5: Sample San Diego Bay Repository environmental data set

6. Expanded data access scenarios

The integration of database, archival storage and Web technology promises to facilitate the manipulation of large data sets and large collections of data sets. One goal is to enable data analysis on terabyte-sized data sets retrieved from petabyte archives, at an access rate of 10 GB/sec. Current supercomputer technology supports a 1 GB/s access rate to 1 terabyte of disk. For a teraflops supercomputer with 10 TB of disk, data rates on the order of 10 GB/s will be feasible. This will require, however, support for parallel I/O streams, and support for striping data sets across multiple peripherals. Fortunately, the software technology to support third party transport of data sets across parallel I/O streams is being developed in the HPSS archival storage system (Coyne [9], Watson [10]). Data redistribution mechanisms for the parallel data streams are being standardized as part of the MPI-IO (Snir [11, 12]) effort. The expectation is that the initial usage prototypes described above can be extended to support supercomputer applications that analyze arbitrarily large data sets.

A second goal is to provide ubiquitous access to scientific data sets. Scientific applications should be able to access data and cache it locally no matter where the data is originally located. Some of the key requirements of such a system are:

• **heterogeneous data sources:** Possible sources for data include databases, archives, file systems, and anonymous FTP servers on the Web. An API is needed that will allow an application to specify a data source, establish a connection, select a data set based on requested attributes, and then cache the data set locally.

• **parallel I/O:** Because of the size and number of data sets that can be accessed for analysis, mechanisms for redistribution of data sets from multiple peripherals onto parallel compute nodes are needed. The emerging MPI-IO standard will be the foundation for the API we are constructing.

• **distributed computation support:** Data sets may be distributed to multiple platforms, for analysis by methods that are retrieved from ORDBMS. Support for distribution of computation objects is needed.

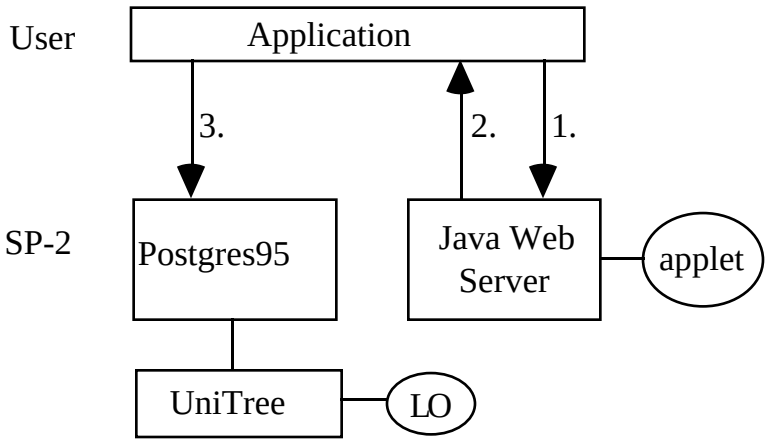
• **third-party data access:** If both data sets and computational methods are distributed to a remote platform, mechanisms are needed to allow the method to access a temporarily cached data set.

• **third-party authentication:** Similarly, methods and data sets need to validate their interoperation through an authentication mechanism that is independent of the local operating system.

The end result is a data handling environment where the focus is on moving and caching data rather than moving and distributing applications. The operating system at each server or compute platform controls use of the local resources. The data handling environment provides a higher-level infrastructure that supports remote access, authentication, and data movement.

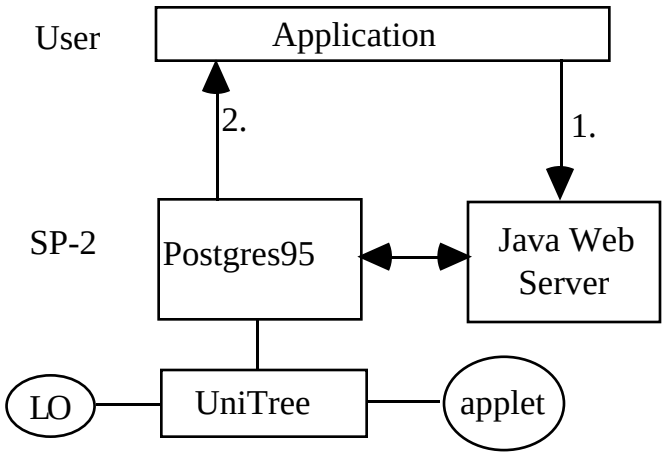
An example of this environment is shown in **Scenario I**. A user makes a request of a remote system for a particular data set. The process consists of retrieval of an applet stored on the local disk of the system, which is then used to access an ORDBMS database. The data object is retrieved from the archive that is linked to the database.

A prototype of this system based on a Java interface to the Postgres95 ORDBMS is being developed. Interfacing Java to Postgres95 required porting the Postgres95 client interface library to Java. This enables a Web client that has Java capabilities to interact directly with the Postgres95 DBMS. Part of this work has been inspired by a prototype developed by John Kelly at the Blackdown site (ftp: // substance.blackdown.org / pub / Java / Java-Postgres95). In particular, we have added support for large objects, a functionality that was not provided earlier.



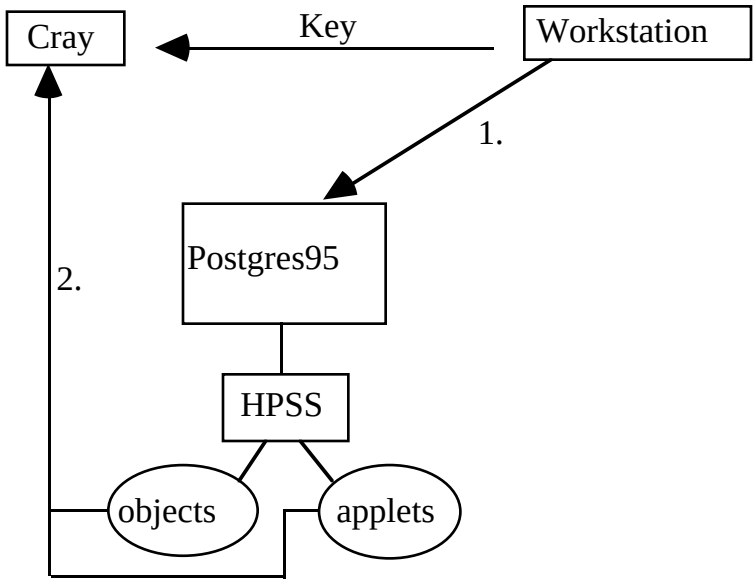
Scenario I: Pulling out applets and large object data

An improvement to this architecture is shown in **Scenario II**. The applet is stored as a method within the ORDBMS. The request to the Web server results in the extraction of the applet out of the archival storage system, and its transmission to the remote user. The applet is then executed on the remote system to access data objects through the ORDBMS system.



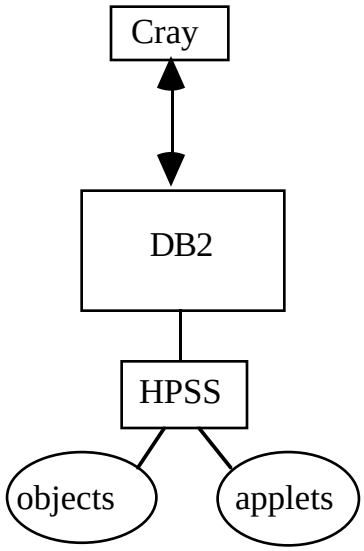
Scenario II: Pulling out large object applets

A further extension of the system is shown in **Scenario III**. The Web server interface is directly integrated into the ORDBMS. A request for analysis of a data object results in both the data object and the associated method being moved to a compute platform. To provide data privacy, the data set may be encrypted. The encryption key is sent to the method, thus providing both third-party authentication and mechanisms for controlling third-party data access.



Scenario III: Third-party data access and third-party authentication

Finally, as shown in **Scenario IV**, the above capabilities can be implemented directly within I/O libraries that are used by a scientific application. The application then directly accesses the remote database/archival storage system to retrieve a data set. Data subsetting and redistribution can be provided by application of appropriate methods to the data set on the compute platform which supports the ORDBMS.



Scenario IV: Supercomputer analysis of scientific data sets

Acknowledgments

This work was supported in part by DARPA grant F19628-95-C-0194 on Massive Data Analysis Systems, and in part by the NSF cooperative agreement ASC-8902827 for the San Diego Supercomputer Center.

References

1. R. W. Moore, "High Performance Data Assimilation," Proceedings of the Committee on Information and Communications R&D (CIC) of the National Science and Technology Council, July, 1995. http://www.sdsc.edu/EnablingTech/InfoServers/HPDA.html

2. R. W. Moore, "Distributed Database Performance," SDSC Report GA-A20776, (December 1991).

3. The Design of a Parallel Data Handling System for Scientific Data Management and Mining, Reagan W. Moore, Richard Frost, Mike Wan, Joe Lopez, Richard Marciano. Submitted to PDIS, December 1996, Daytona Beach, Florida. http:// www.sdsc.edu / EnablingTech / InfoServers / parallel-mining.html

4. Illustra - Illustra Information Technologies Inc, "Illustra User's Guide",1995.

5. M. Stonebraker et al., "The Implementation of Postgres", IEEE Transactions on Knowledge and Data Engineering (March 1990).

6. M. Stonebraker and G. Kemnitz, "The POSTGRES Next-Generation Database Management System," Communications of the ACM, 34 (10), 78-92 (October 1991).

7. Archival Storage Research at SDSC, http://www.sdsc.edu/EnablingTech/archstor.html

8. Richard Marciano, "High Performance Computing Web-Based Simulation Environments", High Performance Computing 96, New Orleans, LA, April 8-11, 1996.

9. R. A. Coyne, H. Hulen, and R. W. Watson, "The High Performance Storage System," Proc. Supercomputing 93, Portland, IEEE Computer Society Press (November 1993).

10. R.W. Watson and R.A. Coyne, "The Parallel I/O Architecture of the High-Performance Storage System (HPSS)", the 1995 IEEE MSS Symposium.

11. M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra. MPI: The Complete Reference. MIT Press, 1995.

12. Marc Snir, Peter Corbett, Dror Feitelson, and Jean-Pierre Prost. Draft Document for a Parallel MPI IO Library. (Draft document presented for informal consideration in MPI-2 standardization process), January 14 1994.